

AN INTRODUCTION TO
YOUR NEW PET™
(REVISED)

 commodore

PET™
2001 Series

personal
computer

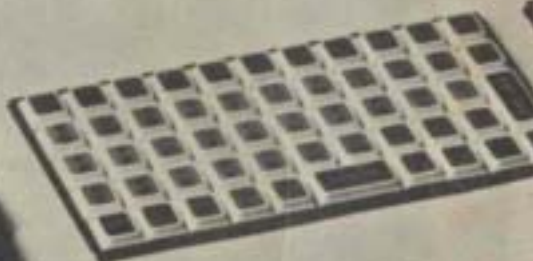


Table of Contents

CONGRATULATIONS	1
UNPACKING YOUR PET AND TURNING IT ON	1
TOURING THE KEYBOARD	2
Figure 1. The keyboard	3
Figure 2. The characters without shifting	3
Figure 3. The characters with the SHIFT key in use	4
<i>Exercise 1 - Testing the keyboard</i>	4
<i>Exercise 2 - Using the cursor (... and introduction to screen editing)</i>	6
<i>Exercise 3 - Using graphics</i>	7
Figure 4. Rocket drawing using the graphics keys	7
Figure 5. Graphics keys used to draw rocket	8
<i>Exercise 4 - Creating a program</i>	8
<i>Exercise 5 - Listing and running your program</i>	10
<i>Exercise 6 - A mending the program</i>	10
<i>Exercise 7 - Screen editing</i>	12
<i>Exercise 8 - Using the reverse field</i>	15
<i>Exercise 9 - Programming cursor movement</i>	16
EDITING; A REVIEW	17
Direct Cursor Control	17
Programmed Cursor Control	19
Programmed Edit Functions	20
EXPLORING BASIC	21
<i>Exercise 10 - Printing on the screen</i>	22
<i>Exercise 11 - Using the built-in clock</i>	22
<i>Exercise 12 - Setting the clock</i>	23
<i>Exercise 13 - Solving mathematical problems</i>	23
<i>Exercise 14 - Animating your PET</i>	30

Congratulations ...

We congratulate you on your purchase of a Commodore PET computer and welcome you to the growing legion of PET users.

The number of applications for the PET is as great as your imagination and the number of programs available.

We shall attempt, in this modest booklet, to introduce you to the art of doing your own programming and to the PET method of using prepared programs.

We do recommend you to the brief bibliography in this booklet's appendix. Enjoy learning about your PET and the world of computers as much as this booklet's authors did.

By the time you're through with this booklet you should be on your way to enjoying and using your PET to its fullest.

Unpacking your PET and turning it on

Please check the carton for any special unpacking instructions.

And examine carefully your P17 for any concealed damage. If there is any, report this IMMEDIATELY to both ('commodore (or your dealer) and to the shipping agent.

Remove your PET from its protective shipping carton, and place it on the counter, desk or other suitable surface, then plug it into any standard, grounded electrical outlet.

Push the rocker switch, in the lower left rear where the line cord enters the unit, to the power-on position.

Momentarily you may be able to see that the PET television display contains a collection of random letters and symbols. This is normal. On "Power up," PET's electrical circuits have to wake up slowly before they can function and clear the screen.

Almost as fast as you can blink an eye, the screen will clear and one of the two messages below will be printed out in white letters on the black screen. The second line of the message will vary depending on

USING YOUR CASSETTE TO SAVE A PROGRAM	31
USING YOUR CASSETTE TO LOAD A PROGRAM	33
APPENDIX	
1. Error messages	36
2. Basic commands	40
String functions	43
Arithmetic functions	44
Special symbols, commands and statements	45
I/O Commands	46
3. Special keys	46
4. Cleaning Your PET	47
5. Cleaning and demagnetizing your tape deck head	48
Figure 6. Cassette deck head area	49
6. References	49

which memory option of PET you are using. It tells you how much memory space is free for you to use.

4k power-on display



8k power-on display



(A byte is the fundamental data element of the PET computer and corresponds roughly to one letter or digit of information. For the curious: the 4k model should show - in theory

"4096 bytes" and the 8k, "8192 bytes. "But a few hundred are used" by the PET internally. The balance shown "3071 and "7167" are net available bytes.)

If you fail to get the power-up display the first time, try turning the power switch *slowly* off then back on. Rarely will PET fail to respond to this, but if it does, turn to the *Hints if You Have a Problem* section at the back of this pamphlet.

Most noticeable in the display is a flashing white square called a cursor. Whenever PET is waiting for some keyboard information, the cursor will begin blinking and this is where the next character will appear if it is typed in.

Touring the keyboard

But, before you can speak to your PET, we need to take a brief tour of the keyboard.

Each key has a thin, transparent plastic film covering the keytop which should be removed. This protection was left in place to protect the keys against scratches during shipping. To remove the film, carefully peel off with your fingernail.

Key top legends bear much resemblance to those of a standard typewriter keyboard, but there are a few differences.

The letters are all in virtually the same place as on a standard typewriter keyboard, but, for your convenience in numerical computations, the numbers are separate and laid out very much like a calculator keyboard. (See Figure 1.)



Figure 1. The keyboard

On most typewriters, if you strike a letter key, without shifting, you will get a lower case letter. On your PET, if you press a letter key without shifting, you will get the capital letter. (See Figure 2.)

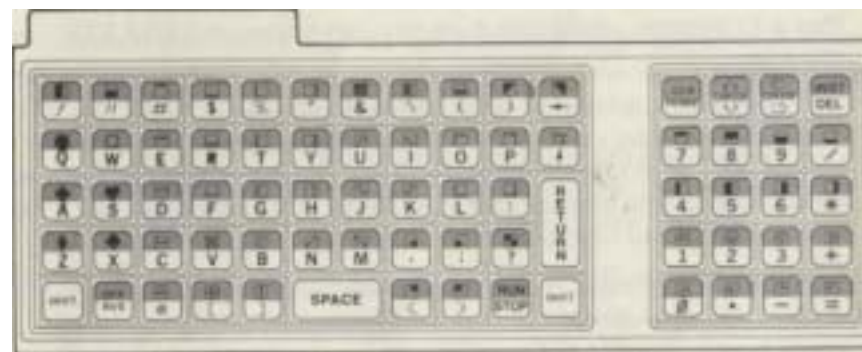


Figure 2. The characters without shifting

If, on the other hand, you simultaneously press the shift key and a letter key, you will get the particular graphic that appears above the letter: (See Figure 3.)

The graphics characters are a special set of symbols unique to PET. They are used to draw pictures and lines on the screen and to perform simple animation. The graphics can be printed on the screen just like any other letter or digit.

For now, locate the **L** key and press it a number of times to get a row of characters-AAAAA -on the screen. (Do *not* use the **SHIFT** key. If you did, you'd get **L**.)

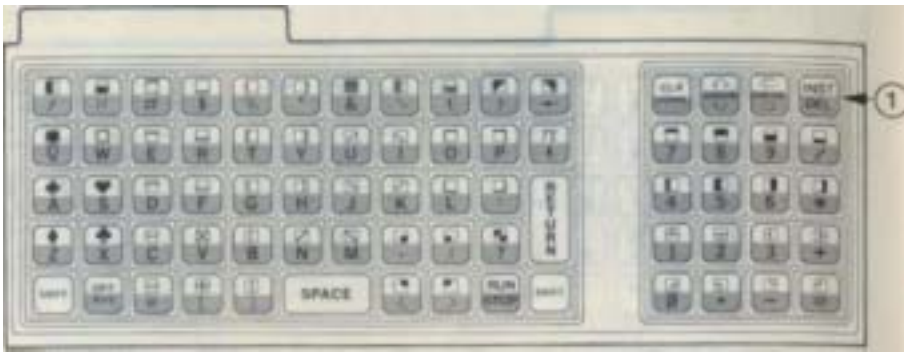


Figure 3. The characters with the **SHIFT** key in use

Next, press the **INST DEL** key ^{labeled} ① on the keyboard illustration (above). Type a different letter. Then press **INST DEL** again. Did you see the character erase?

(P' to again. without shifting, you're getting DELETE. Shifting would get you INSERT.)

Play a little game where you type in more letters and DELETE them too.

Remember that no matter what key you press, *there is no way to damage the insides of your PET by normal keyboard operation.* (Of course, PET is not intended to survive hard falls or attacks with sharp objects - but with normal care it will give you years of service.) Do not ever be afraid to experiment.

Test out the keyboard by trying the following sequence of key-strokes. Don't worry about making typing mistakes; you already know how to correct them.

Exercise 1 - Testing the keyboard

H I SPACE P E T RETURN

The **RETURN** key is a special signal to PET that you have finished typing a line and it should do something with it. This feature allows you to edit the line and get it correctly typed before your PET can act on it.

The important thing about this exercise is to get the following display on the screen after you've done:



Try it again if you wish. PET is just telling you that it does not understand what you said. PET speaks a language called BASIC which was invented by some people at Dartmouth University especially for making the resources of a computer quickly and easily available to those with no previous experience. "Syntax" is, of course, the same word you encountered when studying grammar; it refers to the rules of language. So, "Syntax Error" means you haven't followed the rules exactly. And, in BASIC, *you must be exact.*

Exercise 2 — Using the cursor (... and introduction to screen editing)

By this time you have probably found that (if you are not a typist — or even if you are) it's sometimes challenging, to say the least, to type lines into your PET correctly the first time. Sure, you can use

the **INST DEL** key to erase the last character typed, as we explained earlier. But what if you typed something wrong at the beginning of the line? You could delete characters back to that point, then retype.

But there is an easier way: PET has a feature called *screen edit* which allows you to move the cursor to any position on the line and at that point either insert, delete, or retype.

(The movement of the cursor is non-destructive to the characters over which it passes. The characters will not be deleted or changed as you move the cursor around on the screen.)

Locate this row of keys on top of the right hand numeric keypad.



These are all double function keys, their action depending on

whether or not a **SHIFT** key is used. Press the labeled CLR HOME

key top. See the cursor move to the top left corner of the screen? This is the "HOME" position.



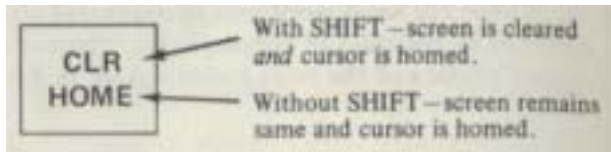
 www.commodore.ca

Free for personal use but you must have written permission to reproduce

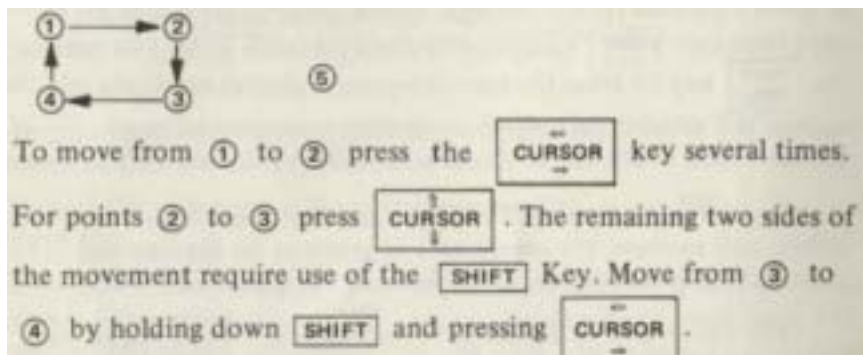
The same key pressed while the **SHIFT** key is pressed clears the screen. Hold down **SHIFT** this time and then press **CLR HOME**

If there were any characters on the screen, then they were all erased ... or "cleared."

Both functions of this key affect the screen.



The best exercise to learn the individual cursor movement keys is to move the cursor right, down, left, and up in a sort of circle path to return to the original starting position. You will move the cursor on the screen in a path like this:



(If you press the last key too many times and wind up in position you have discovered another feature called "wrap-around" which has moved the cursor to the end of the previous line. Type

without the shift key held down to move the cursor

back to position ④.

The home stretch from ④ to ① is easy. You can either hold down

SHIFT and type **CURSOR** repeatedly until the cursor is in position 1 or type **CLR HOME** once to move the cursor "HOME." Try it both

ways. Try moving the cursor around the screen between two arbitrary points. Practice until you are confident you can put the cursor where you want it on the screen.

Exercise 3 - Using graphics

If you have accomplished moving the cursor, then you can use your PET like an electronic sketch pad. The characters on the upper half of each keytop are called graphics. When you hold down the **SHIFT** key as you type, the graphics are printed instead of letters or numbers.

Now let's draw a figure that should look very much like this by the time we get through.



Figure 4. Rocket drawing using the graphics keys

Follow the instructions exactly as shown in the diagram that follows:

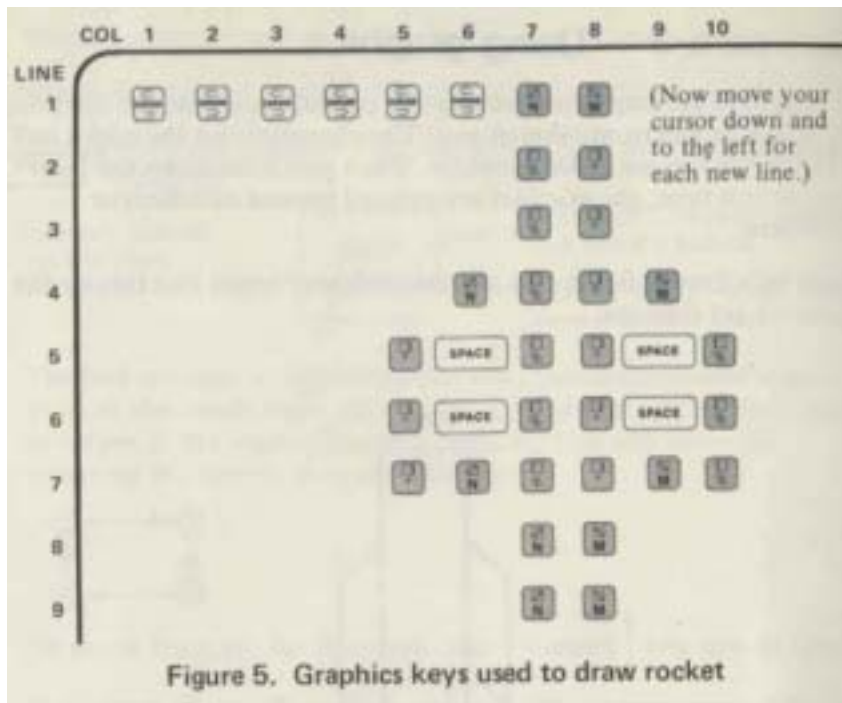
Remember the **CLR HOME** and **SHIFT** keys? Use these keys now

to clear the screen.

Move the cursor to the right 6 spaces; as shown in the diagram.

Press **SHIFT** and type the graphics. Now you use the cursor keys to get the cursor in position to type the next line.

NOTE: Shaded keys are keys that must be accompanied by pressing the **SHIFT** key.



.VOTE: Do **not** press **RETURN** at an t^ time in this exercise. Your PT,-Twill think you've,finished, it will not understand and



If this happens. first clear the screen again and start over.

When your rocket is complete, move the cursor to the left edge of the screen at line 10.

Now type **N E W** **RETURN**

The Cursor should be in the left part of your screen when you've done all the above.

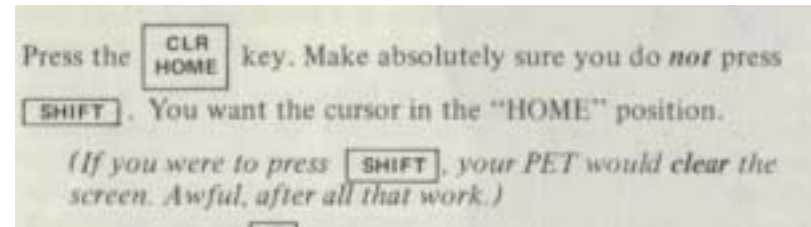
Exercise 4 - Creating a program

When you have finished this exercise, you will have drawn a picture on the screen. You probably went to a lot of work to create this picture. You'd like to preserve it so you can view it again.

So let's turn each line of the picture into a program step and see what happens.

The importance of a program to a computer can be likened to the importance of a driver to a car. The car does nothing without a driver and the computer does nothing without a program.

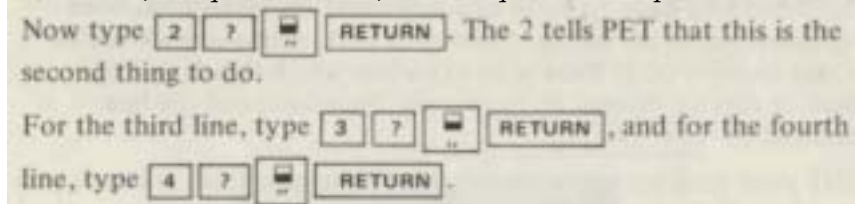
A program is stored as a list of steps or instructions in PET's memory. Before we can create a program in its memory, we should make PET forget about any previous program. This is what we did when we typed the word NEW. Use the command any time You want to enter a ,NEW program.



Now type **1** **?** **RETURN** The number 1 tells PET "This is the first thing to do." The ? tells PET to print, and the quotes tell PET to print a MESSAGE.

(If you make an error, do not try to correct it. Instead

press **RETURN** , then move the cursor up and type the correct number, the question mark, and the quotes. Then press **RETURN**



Notice that the only thing that changed has been the number (1 , 2, 3 and 4) that tells PET "this is the th thing to do." So now tell PET the 5th, 6th, 7th, 8th, and 9th things to do, just like we've done with the first four. Be sure to keep the numbers in the right sequence.

Stop when you reach the line containing the word "NEW," because you don't want that word included in your program. Using the



key, move the cursor down the screen until it is below

the word "READY." Would you believe you've just created a computer program?

Exercise 5 - Listing and running your program

Clear the screen and type:

LIST

RETURN

LIST is a command to your PET to print the lines of program stored in memory onto the screen so that you can look at them. You should see something like this on your screen



The ? that you have typed in as a shorthand for PRINT has been expanded out in the listing. Other than that, everything should be as you typed it in. If there is an extra line which should not be there, it may be deleted by typing just the number of the line

followed by

RETURN

LIST your program again if you wish. When everything is just as you want it to be type RUN

RETURN

There! Your picture will appear on the screen. RUN tells PET to execute the BASIC program you have entered, starting at the lowest line number step and proceeding with subsequent steps in ascending line number order.

Exercise 6-Amending the program

RUN your program again. If you did not clear the screen first, you may have seen the old rocket disappear at the top of the screen and the new rocket roll up from the bottom of the screen.

This phenomenon is called "scrolling." When PET is printing in the bottom-most line of the screen, everything moves up rather than the

cursor moving to a lower line. PET cannot scroll the other way, however. Information that scrolls off the top of the screen is lost.

We can use this scrolling effect to our advantage to produce an animation in which it appears as though a stream of rockets are blasting off from the bottom of the screen and are streaking off the top. To do this we will learn a new BASIC language command.

Type this line in

100 SPACE GOTO SPACE 10

The line number (100) was chosen so that it would be greater than any you had used previously and thus would be the last step of your program to execute.


GOTO is a BASIC command to break the sequential execution of statements and "go to" the line number specified. If you entered the rocket picture with line numbers exactly as shown, line 1 is the first line of the program which prints the rocket picture. Change the target line of the "go to" to correspond to your first line number if it is not line 1. The effect of line 100 is to repeatedly print the rocket and scroll it off the screen.

But, because we don't want the rockets to be touching nose to tail, we'd like to add some space between them. When we typed LIST, we noticed that the last line number was 9. (We've since added line 100).

Any numbers greater than 9 and smaller than 100 will be positioned correctly in line number sequence by your PET. So let's add the


statements:



Now type RUN. As soon as you press  PET will execute your program.




Rockets should be flashing on the screen so fast that it may be difficult to see them. The speed at which characters are printed on the screen can be controlled while the program is running by

pushing the key . Hold this key down while you watch the

screen. Now, release the key. Use of this key reduces the printing speed to about 2 lines per second.

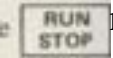
The program you have created contains what is called an "infinite loop." Statement 100 does not contain a condition to stop running the program and cease printing the rocket, but unconditionally goes to the start of the program over and over. It will continue like this forever unless you pull the plug.

*(Pulling the plug or shutting **off** the power-on switch not only stops the program, it also destroys the program statements. You've put in a lot of time typing theta and stay not want them destroyed.)*

PET has a key to press:  This is a STOP function when you do **not** press the **SHIFT** key. This will effectively "pull the plug on this program, without losing the program statements.

PET will respond with something like:

BREAK IN LINE 8

This message means that execution of your program was stopped when it reached line R (or whatever line it was in your case) because you pressed the  key

At this point you may want to save your program. See page 31, "Using your Cassette to Save a Program."

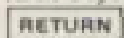
Exercise 7 - Screen editing

One of the handiest features of PET is the ability to modify easily the program you have entered, as we have just seen.

You can change a single character or you can add characters to lines you already have. You can see exactly what you are changing because the changes are visible as you enter them.

Let's try it.


*(But before we start a new program, let's type **new** and press*

 *I titts is important: it clears aft previous programs in your PET and thus avoids any confusion.)*

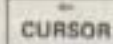

Type in: **10 PRINT "HELLO, HOW HOW ARE YOU?"** 

We have one too many HOWs in the line. Let's type LIST so we can see the line on the screen.




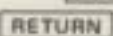
Now press  and  together. Repeat.

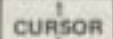
This will move the cursor *tip* two lines from its lower position to the first position of line 10.

Now press  (without the  key) several times until it

is over the space after the W in either one of the HOWs.

Now press  4 times. The extra HOW and a space are gone!

Press  to tell your PET you've finished editing this line.

Now press  to get to a blank line on the screen. Type LIST.



You see how easy that was.

Do the same thing will have eliminated the HOW:



Now let's insert the missing HOW.

Press **SHIFT** and **CURSOR** together. Repeat as before.

Now position the cursor over the A in ARE by pressing several times.

Now press **SHIFT** and **INST DEL** then H
SHIFT and **INST DEL** then O
SHIFT and **INST DEL** then W
SHIFT and **INST DEL** then **SPACE** and **RETURN**

then **CURSOR** to get past the READY in the display. Type LIST.

You

10 PRINT "HELLO, HOW ARE YOU?"
READY.

With editing that easy you need have no fear of making typing errors. Agreed?

Let's try another interesting screen edit. First, type NEW to clear out the old program.

Enter:

1 0 P R I N T "ANYTHING"

RETURN

(This time we won't type LIST each time we make a correction.)

Press **SHIFT** and **CURSOR** so that you position the cursor over 1 in line # 10.

Press 2 then press **CURSOR** until it reaches the A in ANYTHING"

Now type EVERYTHING" and press

RETURN

Now press **SHIFT** and **CURSOR** together so that you position the

cursor over the 2 in line #20. Press 3 then **CURSOR** over to the

E in EVERYTHING and type NOTHING". Press **SPACE** three times. (Because EVERYTHING is three letters longer than NOTHING.)

Press **RETURN**

Now type LIST and press **RETURN**

You'll read:

10 PRINT "ANYTHING"
20 PRINT "EVERYTHING"
30 PRINT "NOTHING"

Interesting? Think of the applications. If you want to repeat a complex statement several times in the same program ... or if you want to change just a part of a statement on one line and enter that amended statement on another line.

Exercise 8 - Using the reverse field

Every key on the keyboard, with the exception of a few which we shall note, prints almost exactly what you see onto the screen. We say "almost" because the screen displays characters in white

on a black background. There is a **OFF RVS** key which, when pressed,

causes all subsequent characters to be displayed in reverse field - black on white - on that line.

Type A B C

OFF RVS

A B C and you'll see:

A B C

Your PET displays 128 unique symbols which, with the addition of reverse field, really adds up to a total of 256 different characters that can be displayed.

Reverse field remains in effect until a) you type RETURN or b)

hold down the **SHIFT** and type **OFF RVS**

As an example, type:

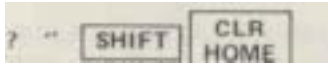
A B **OFF RVS** A B C **SHIFT** **OFF RVS** A B C

You'll see:

A B C A B C

Exercise 9 - Programming cursor movement


Cursor control characters may be programmed into PRINT statements. It is often desirable to clear the PET display under program control. We will do it in a direct statement.




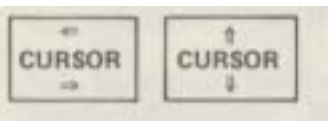
Note that you did not clear the screen by typing these keys, but that the screen



When you have typed an odd number of quote marks you are in this special cursor control character insertion mode.

( Represents a *single* quote mark, for this discussion. And one is an odd number.)


Tilt  is a representation of a CLEAR SCREEN control character. Do not type RETURN yet. Instead type



These print out



which are cursor control characters for CURSOR RIGHT and CURSOR DOWN.





If you now type a second  you will have entered an even number

of quote marks and you will leave the special mode. Typing



will again move the cursor, but this time, without printing anything.

Any time you want to *enter* or *leave* the control character insertion mode you may do one of two things:

1. Enter a second . press  key. Then use your cursor keys to return to the point on the preceding line ... or
2. Delete the first quotation  

Editing: a Review

When you press one of the PET's cursor control keys, you may be in one of two editing modes, as you have already **seen**.



I. DIRECT CURSOR CONTROL

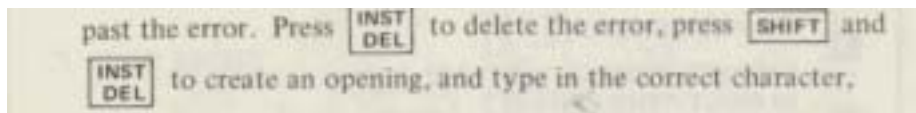
The cursor is moved as soon as you press the cursor control key.




In DIRECT mode, the User is creating program code. The cursor control keys allow the User to insert or delete characters at will unless he specifically indicates (by typing a quotation mark) that the cursor movement is to be a part of the created code.

When entering program code, the User can correct typographical errors in one of four ways.

- A. Delete all characters back to the error, then retype.
- B. If no quotation marks have been used, backspace (cursor left) over the intervening characters until the cursor is positioned over the error, retype the character, then forward space (cursor right) to the next desired character position to be typed.

- C. If a quotation mark has been used, press  to leave the program line. Then move the cursor up  and over to one space



past the error. Press  to delete the error, press  and  to create an opening, and type in the correct character, then forward space to the next desired character position to be typed. Programmed cursor control is no longer in effect.

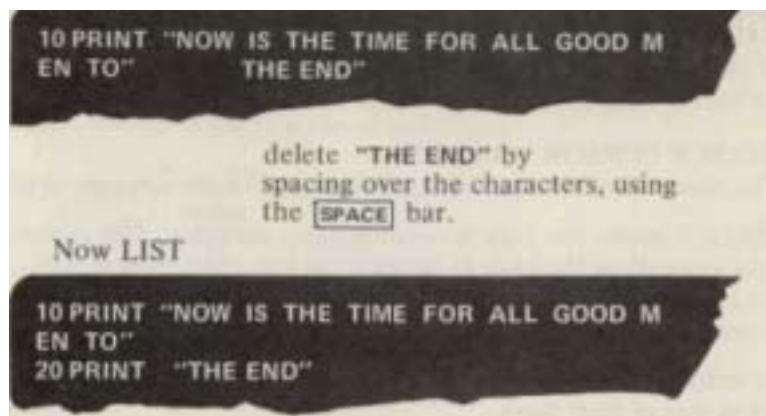
- D. Another method is to close the quotes (type the ending quotation mark) then backspace to the offending character and retype. Again, programmed cursor control is no longer in effect.

There may be occasions when it is appropriate to lengthen a statement line. If the cursor is moved to the end of an existing line, the additional characters may be typed in. The cursor will wrap around to the next lower line if more than 40 positions are used. If the lower line contains a program statement, it can be over-typed. Extra characters remaining from that previously typed line must be deleted or they will be incorporated into the line being edited.



Original Program



Move the cursor until it is, positioned over the closing quotes in statement 10, and type GOOD MEN TO.



If you wish to insert characters within a statement line, position the cursor over the first character to be shifted to the right.



press  with the  key. If the new spaces increase line length to greater than 40 spaces, a space will open up between the line being edited and the next program line, and the characters to the right of the insertion will move into the opened space. This is difficult to show on paper, so just follow the instructions and watch the result on your screen.

I. Type this program

```
10 PRINT "NOW IS THE TIME TO"
20 PRINT "THE END"
```


2. List the program

3. Move the cursor to the letter T in the word TO in statement 10.

4. Hold the  key and press the  key 17 times

(Here's where the screen will show a space being opened between statement lines)

5. Type **FOR ALL GOOD MEN T**

6. Press 

7. LIST the program again

Using Direct Cursor Control while coding a string literal:

To edit a string literal, such as a print message or a data statement, the user must press the **RETURN** key and leave the statement line. A literal cannot be edited (except for character deletion and retyping) while it is being originated, because all cursor controls except delete and insert are programmable. The user must leave the statement line via a carriage return, then move the cursor back to the offending character and retype. Furthermore, to program cursor controls within the string after having left the line, the user must use

the **INSERT** function to open up spaces into which he can then type the appropriate control character.

The user can, of course, close the quotes, and thereby signal PET that he is through with the literal message. However, once the second quote mark has been typed, PET will no longer recognize cursor movement as a part of created code, and the cursor will move according to the Function represented by the key pressed.

II. PROGRAMMED CURSOR CONTROL

The cursor movement is executed during a program run. It is part of a PRINT statement and has been enclosed within quotation marks.

Function

CURSOR UP CURSOR

DOWN CURSOR LEFT

CURSOR RIGHT CLEAR

SCREEN HOME





















CURSOR INSERT

CHARACTER*

DELETE CHARACTER*

REVERSE FIELD

RESET REVERSE'

Keys to Press	ASCII	(Reverse Field) Character
SHIFT 	145	
	17	
SHIFT 	157	
	29	
SHIFT 	147	
	19	
SHIFT 	148	
	20	
	18	
SHIFT 	146	

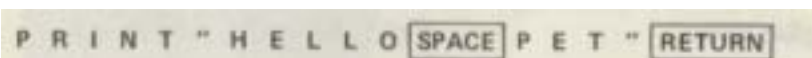
**The INSERT and DELETE functions are not programmable. Use CHRS (20) to delete during program run and CHRS (148) to insert during program run.*

PET uses the quotation mark to signal the beginning of a string literal, as in a DATA or PRINT statement. When attempting to edit a program line, the User should be aware that if PET sees an opening quote, it will consider all cursor movement instructions as part of the string.

The first BASIC command we shall explore will tell your PET to PRINT something on the screen. This is one of the more useful commands, for with it you can make your PET display data, draw pictures, or play games.

Now enter the following by pressing this sequence of keys. (We'll call it "typing" from this point on.)

Exercise 10- Printing on the screen



Did your PET print HELLO PET on the screen? If it did not, then try it again. PRINT is a command which tells your PET what to do with the rest of the line. This example has a message between quotes. The quotes tell your PET to print out the message exactly as it appears within the quotes without any further processing.



Exercise 11 - Using the built-in clock

Now, let us speak to your PET in BASIC and get it to tell you what time it is. Your PET has a built-in clock that starts from 0 the moment you turn on the computer.

To discover the elapsed time, type:



The ? is a shorthand which you may use instead of always typing PRINT when you want your PET to print something. The \$ at the end of the word TIME tells PET to print the time in hours/minutes/

seconds. Though the elapsed time may be different, you should see



The first two digits are elapsed hours, the second two digits are elapsed minutes, and the last two digits are elapsed seconds - which, in the above example, means that PET has been running for 00 hours, 11 minutes, and 30 seconds. The time you see on your PET, however, will depend entirely on how long you have had it on thus far. PET's clock is crystal-controlled and very accurate. It is also a 24 hour clock which means it will count up to 23:59:59 then roll over to 00:00:00.

Exercise 12 - Setting the clock

It is very easy to set your PET clock. Assume it will be 12:30 p.m. in a few seconds. Press the following sequence of keys:

`TIMES="123000"`

When the designated time (12:30 p.m.) comes up on your watch,

press  and PET will set the time.

Substitute your current local time and try setting the clock as in the previous example. (If the time is 9:30, be sure to type "093000" you need 6 digits.)

Now, whenever you type:



your PET will tell you the correct time. Remember that if you turn the power off, the clock will stop running and you will have to reset it when you turn the power on again. Once you have reset it, though, you have a highly accurate built-in clock available at all times. Just

type in `? TIMES`  and there it is.

Exercise 13- Solving mathematical problems

BASIC is essentially an algebraic language which means that you can use your PET much as you would a pocket calculator. Though it

packs the power of several programmable calculators put together, it is as easy to use as a simple four-function calculator. Furthermore, everything you type into it is instantly displayed on the screen, and that makes it easier to keep track of what you are doing.

To perform arithmetic on your PET, simply tell it in BASIC to print your answer. Note how much it looks like a direct question:

? 2 + 2 **RETURN**

When you press the RETURN key. PET prints the result on the screen.



PET arithmetic is not complex or mysterious. But, like human arithmetic, it does have rules. One of the most important rules is the *order* of operation:

- | | |
|--------------------------------|-----------------------------|
| 1. exponentiation | <i>t (above RETURN key)</i> |
| 2. multiplication and division | * / |
| 3. addition and subtraction | + - |

In direct mode, type:

?9*3+7*6-5*4/3*2 PET

will respond: 100.

666667

You (because you're the one in charge here) decide how to group the expression so the result will be correct. You do this with parentheses.

Type: ?(9*8+7)*6-5*4/3*2

PET says: 460.666667

Type: ?(9*8+7*6-5)*(4/3*2)

PET says: 290.666667

Type: ?((9*8+7*6-5*4)/3)*2

PET says: 62.6666667

Type: ?(9*8+7*6-5*(4/3)*2

PET says: 214.666667

PET does all the work inside the parentheses first, before it does anything outside of them. Once inside parentheses, PET does multiplication and division, then addition and subtraction. After all the work inside the parentheses is done, PET moves outside and does everything" there.

Let's look now at some of the arithmetic functions and find out how to use them. First, type NEW, to delete the program we were using. Now, type these program lines:

```
100 FOR I=1 to 10 120
PRINT I,I*I,SQR(I) 130
NEXT I
```

and RUN.

The result represents I (the count or index), I squared (the * means multiply), and the square root of I. SQR(X) is the Square Root function, and finds the square root of whatever number you put inside the parentheses. Like all BASIC functions, SQR(X) requires an *argument* any number you choose, enclosed in parentheses. Arithmetic expressions can be used if you like instead of numbers, like SQR(5*20)

All the arithmetic functions are used in a similar fashion. Turn to the BASIC keywords (Appendix II) and find the section on Arithmetic functions. You can check out the way each one works by substituting it for one of the functions in line 1 2Q of the program. The brief description of the function, along with actually trying it yourself, will show you how to use it.

 www.commodore.ca

Free for personal use but you must have written permission to reproduce

Try using the string functions, too. You'll need a new program to do that (be sure to type in NEW first):

NEW

```
10 FOR I = 1 TO 10
```

```
20 INPUT "ENTER A STRING";A$ 30
```

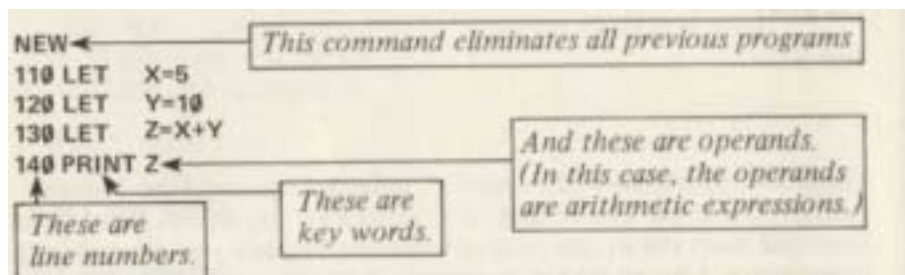
```
PRINT LEN (A$)
```

```
40 NEXT I
```

Lines **10** and **40** you already know about. In Line **30**, the new item is the dollar sign. This tells PET you are entering letters or graphics instead of numbers. Line **30** prints the function we selected for this example. Since you know your name best, use your name as the data you **INPUT** in line **20**. PET should print the number of letters and spaces in your name when you **RUN** the program. PET does count spaces in strings—even if that's almost the only place it wants them!

Now let's go a step beyond.

First type these lines. Note the comments in italics; they explain each piece of the program lines.



Line **110**, **120** and **130** are called "assignment statements", because they are used to assign values to letters of the alphabet. Line **140** is a **PRINT** statement and causes PET to print the value of the letter **Z**. (PET knows the value of **Z** because statement **130** told it that **Z** is the sum of **X** and **Y**.)

Of course, this is a trivial program. You've already learned how to type `5 + 10` and have PET print the answer. But it gets more interesting as we go along.

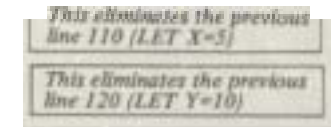
Now, if you **RUN** this program, PET will print:



You, as the programmer, can put any number you like in statements **110** and **120**. You don't have to use **5** and **10**.

It would be cumbersome to have to retype lines **110** and **130** every time you wanted to change the numbers. So, PET's BASIC allows you to change the numbers during program execution, using the **INPUT** keyword. Let's change the program to show you how to **INPUT** data:

```
100 PRINT "ENTER A NUMBER"
110 INPUT X
115 PRINT "ANOTHER NUMBER"
120 INPUT Y
```

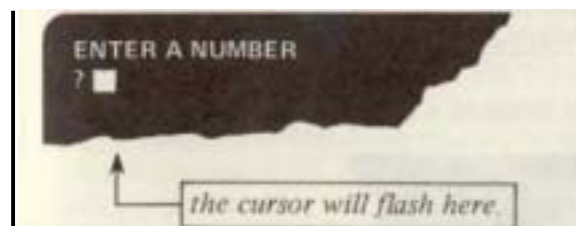


Since lines **130** and **140** stay the same, we won't retype them. Here, line **100** prints a message. When you run the program, PET will print the message so you'll know what you're supposed to do. By using this "*prompt*," anyone can use your program, because he'll be told what to do and won't have to guess. Line **110** will force PET to wait until you type in a number and press **RETURN**.

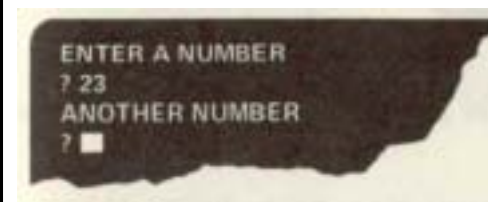
If you press **RETURN** without entering any number, PET will think you don't want to continue running the program. It will jump out of the program and tell you it's **READY** for whatever you want it to do next. Line **115**, like line **100**, prints a message prompting you to enter another number, and line **120** makes PET wait for you to do so.

Now **RUN** the program.

PET will show:

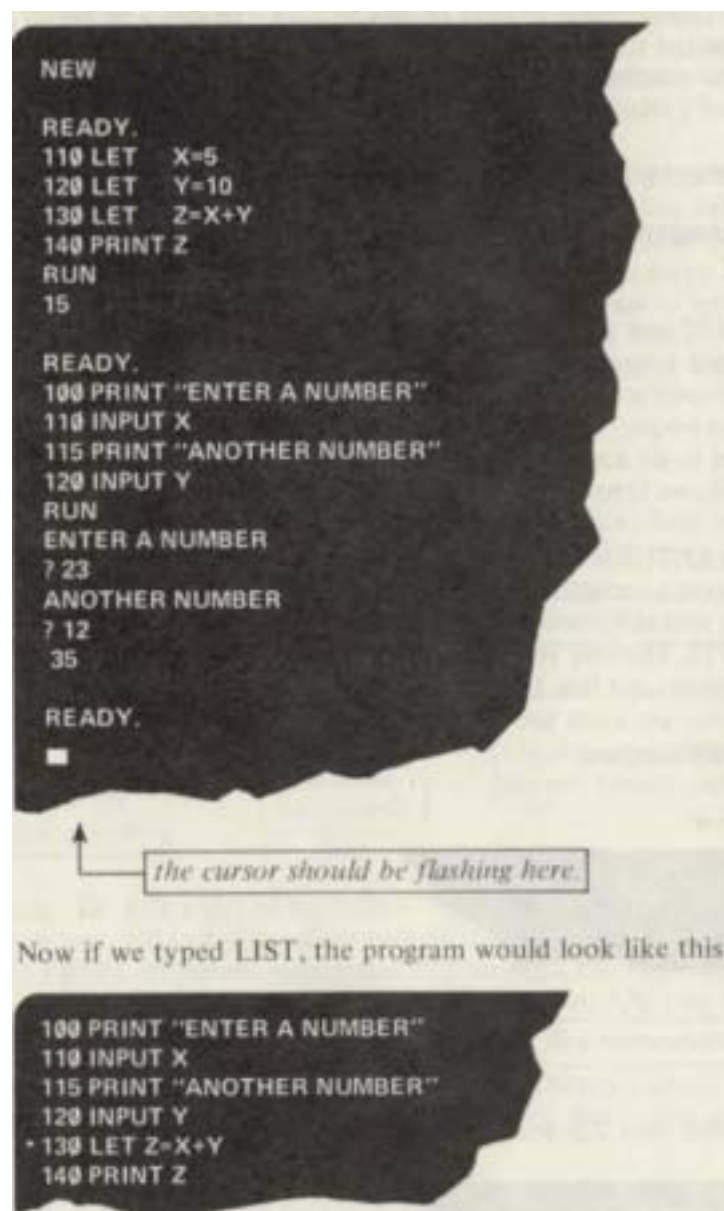


Type a number—say **23**—and press **RETURN**. PET prints:



the cursor will flash here'.

Type another number--perhaps 1 " and press RETURN. PET will print the answer 35 and say READY. Your screen should look like this:



Note that your PET automatically places the line numbers in ascending order, too.

Using INPUT makes it easier to check the numbers, doesn't it? Now you just type RUN, and each time you do so, you can use different

numbers. But if you have 10 pairs of numbers, typing RUN each time can STILL be tedious!

So, BASIC has two features which allow you to do an operation (in this case, add a pair of numbers) as many times as you like without typing RUN. One of these features is the GOTO statement. It forces PET to GO TO a line you specify, instead of doing whatever it would normally do. (In this case, PET would normally stop and print READY when it finishes the sum.) You can do that with this statement:

```
150 GOTO 100
```

Now PET will print the sum, then go back to line 100 and print the "ENTER A NUMBER" message. Try it. Add up several pairs of numbers. When you're ready to go on, just press RETURN without having given PET any data (numbers) and it will jump out of your program and alert you to tell it what to do next. (You'll see PET's READY message and the flashing cursor.)

The second feature is called a FOR-NEXT loop. It allows PET to perform an operation (or a sequence of operations) FOR as many times as you like. The word NEXT is the last line of the sequence, and tells PET that it has completed all the repeatable operations.

Type these lines.

```
90 FOR I = 1 TO 10
150 NEXT I
```

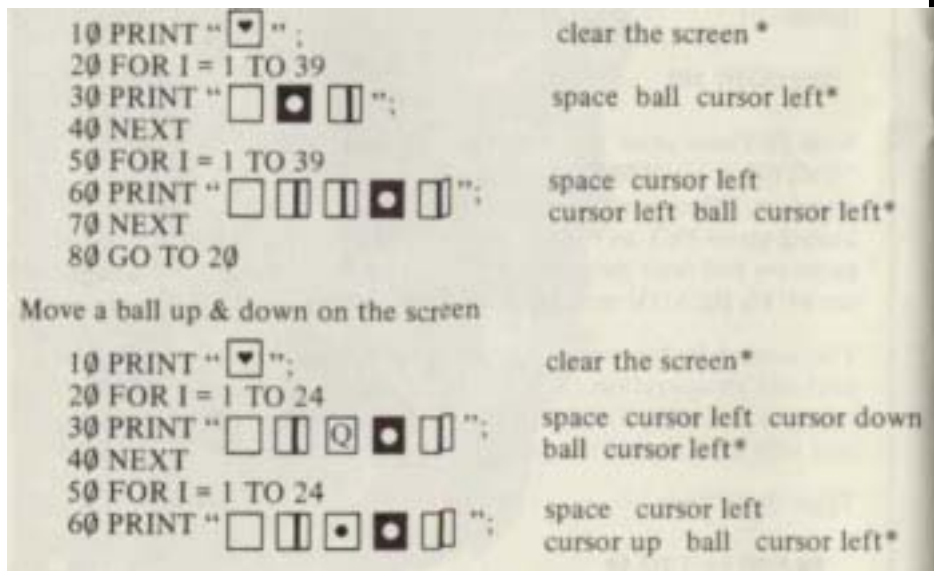
In this example, I is called the "index." PET keeps track of the number of times it performs the sequence of operations (lines 100 through 140) and keeps the count in the index. You set the index to I in statement 90 and tell PET to count to 10. Everytime PET reaches statement 150 it increments, or adds to, the count, and goes back to statement 90. Then PET checks to see if the number in the index is greater than the number of repeats you wanted. If it is, then PET looks for something else to do. If the count is less than or equal to the number of repeats (in this case, 10), it performs the whole sequence again.

In brief, PET will let you enter a pair of numbers FOR as many times as you request. And PET will do as much as you ask to each pair of numbers before going after the NEXT pair.

Exercise 14 - Animating your PET

It's easy to move an object smoothly across the CRT, thanks to PET's programmable cursor controls. The listings below give you the fundamental right-left-up-down motions. Later we will show you how to program an unidentified flying object.

Move a ball right & left across the screen. (Note: all symbols are shown as they appear on screen.)



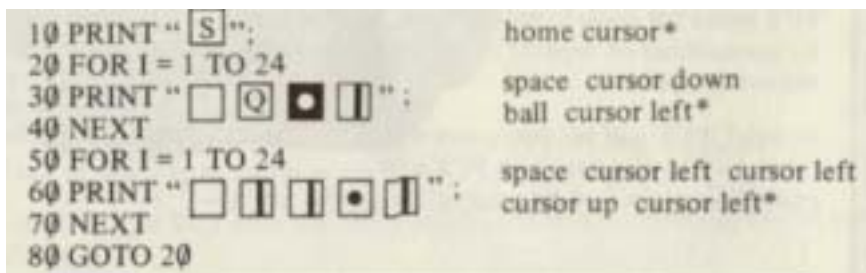
Note: While "cursor up" character and "ball" character (the shifted "Q" character) look alike, they obviously perform quite different functions as you will perceive ... provided you press the right key.

```

70 NEXT
80 GOTO 20

```

And, of course, moving a ball diagonally across the screen (top left to bottom right)



'filter you type, in sequence, a quotation mark and a cursor key, you'll see a symbol appear to indicate the cursor movement or action. This is a guide to these symbols:

PET	Shows	Key guide
reverse field heart	▼	clear screen
reverse field right bracket]	cursor right
reverse field dot	•	cursor up
ball	■	shifted Q graphic
reverse field capital Q	Q	cursor down
reverse field capital S	S	home cursor
reverse field square with line graphic	□	cursor left
space		space

Using your cassette

to save a program

After the effort of entering and debugging your BASIC program, you may want to save it on cassette tape for later use.

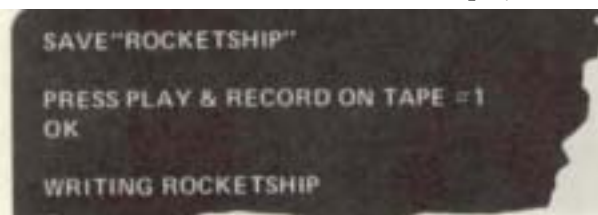
Place a blank cassette in the recorder unit, and press **"REW"** to be sure the tape is fully rewound. When rewinding is complete, press **"STOP"** on the cassette unit. You are ready now to save your program.

It's a good idea to save your program by name. Having completed the rocketship drawing and converted it to a program, let's save it, using the name **"ROCKETSHIP."**

Type **SAVE "ROCKETSHIP"** and pass **RETURN**. PET will display:



Press the correct buttons, and the display reads:



Now, when PET is ready, and the cursor flashes, you should rewind the tape and verify that your program really did get saved. When the tape is fully rewound tune the word **VERIFY** and press **RETURN**

PET will tell you which cassette unit button to press. Do so, and PET will assure you it is **VERIFYING**. Then, when PET is **READY**, you now can be sure your tape has an accurate copy of your program.

If you see **VERIFY ERROR**, rewind your tape and try it again. If you still get **VERIFY ERROR**, save your program and start over. (Note: be sure to use music-quality low noise cassette tapes.)

You can use any name you like for your program. Because your PET will only display 16 of the characters used in a program name, you'll probably want to keep the names short.

When you name your programs, remember that PET doesn't need to have the whole name typed on the keyboard in order to find it, when you ask PET to load it. If you name your program "**ROCKETSHIP**", you may ask PET to **LOAD-ROCK**" and it will find the right program and load it. This means you must be careful to avoid naming one program "**ROCK**" and another program on the same tape "**ROCKET**". PET won't know which is which, and will load the first program with the letters ROCK in its name.

You do not need to incorporate the program name into the program itself. PET saves the program name in a file header when it saves the program, and that the only identification it looks for.

This means you can save your program in small pieces as you write it. You may find it worthwhile to **SAVE** your work every 20 to 30 lines or so, to avoid losing it in event of a power failure. But, since you may not want to type the full program name each time, PET is designed not to REQUIRE: a name:

Type **SAVE** and press . PET will display



and_ when you do so. PET will show



And as soon as the program is saved, PET will add the word **READY** to the display.

You may want to save your program more than once on the same tape to be sure you have a good *copy*. As cassette tapes age, they frequently stretch or wrinkle, especially at the ends. A second, or



even a third **SAVE** may prevent loss of your program and will prove to be worth the extra few minutes it takes to save and verify the extra copies.

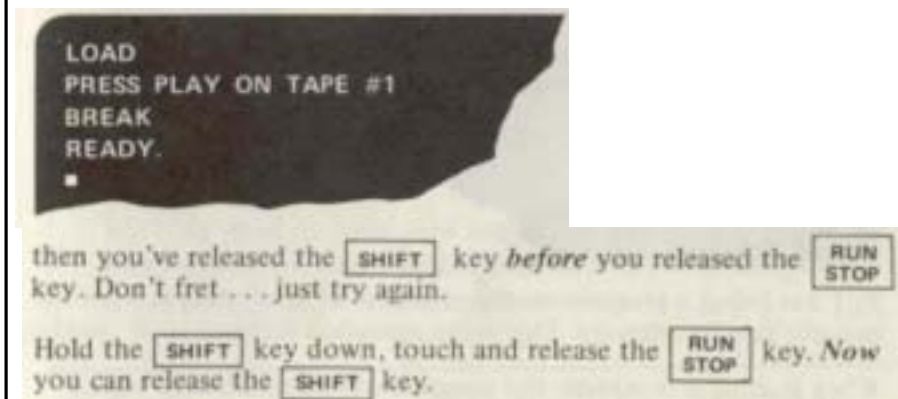
Using your cassette to load a program

The built-in cassette drive in your PET computer is your easy access to a library of BASIC programs, either created by you or purchased from the extensive COMMODORE library.

PET is like a pocket calculator in that it forgets everything when you turn the power off. (Remember what happens to time) That is why PET has a built-in cassette drive. Programs can be saved on tape before power is turned off. They may be restored to PET's memory when power is turned on again.

Take a cassette, open the cover, and place the cassette in just as you would a normal audio cassette. Do not push any cassette keys at [i]is time.

Now, hold down the  key and touch and release the  key. If you see:






 www.commodore.ca

Free for personal use but you must have written permission to reproduce

If you've done all this correctly you should see:

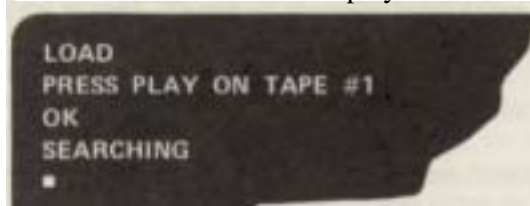


Pressing the  and the  keys caused the command LOAD to be typed on the screen and PET responded by asking you to operate the cassette.

(If you wish, you can also tell your PET to load the program by typing in LOAD and [ instead of the above procedure.)

Press the key labeled "PLAY" on the cassette unit.

Your screen should now display:



NO

This means your pressing of the key is acknowledged and PET is now searching for data on the tape. In a few seconds (about 5-10) you will see:



PET has found a program on the tape and is transferring it from the cassette into its memory. This is the operation referred to as "loading."

When loading is complete, the program will automatically begin executing. Also, the cassette motor will be turned off.

(We're assuming that your PET's program is labeled "PROGRAM" for the sake of this example. It could be labeled virtually anything else.)

Disclaimer on Software:

The complex and extensive software of the PET computer has been thoroughly tested and is believed to be quite reliable. However, no responsibility is assumed by Commodore or your sales agent for inaccuracies.

Commodore Invites You To Submit Your Best Programs

Commodore intends to publish a library of programs for your PET computer. This endeavor has already begun.

We'd like to encourage PET users to submit original programs to us. Those programs that are accepted will earn a royalty for their developers.

If you're interested in writing programs for distribution, write to:

*Software Manager
Commodore Business Machines
901 California Avenue Palo
Alto, CA 94304*

We'll be glad to send you suggestions on programming techniques, procedures for submitting programs, and some guidance in selecting programs.

 www.commodore.ca

Free for personal use but you must have written permission to reproduce

Appendix

1. Error messages

When an error occurs, PET returns to Command level and displays READY on its TV screen. Variable values and the program text remain intact, but the program cannot be continued using the CONT command. GOSUB and all FOR ... NEXT context is lost, insofar as the current run is concerned.

When an error occurs in a program statement, the error message display will indicate the line number in which the error occurred.

When the error occurs in a direct, or command level, statement, no line number is displayed with the error message.

Error Message	What caused the error and how to fix it
---------------	---

CAN'T CONTINUE	Attempt to continue a program when none exists. an error occurred, after a new line was typed into the program, or a correction was made to an existing line.
-----------------------	---

Correct the error, then use a directed GOTO to get back into the program, or type RUN and start over.

DIVISION BY ZERO	Dividing by zero is an error. Check the expression used for the denominator in the offending arithmetic statement. then correct it so it can never be evaluated as 0.
-------------------------	---

ILLEGAL DIRECT	Use of an INPUT, GET, or DEF statement as a direct command.
-----------------------	---

Avoid using these statements as direct commands.

ILLEGAL QUANTITY	The parameter passed to a math or string function was out of range. "ILLEGAL QUANTITY" errors can occur due to:
-------------------------	---

- a. a negative matrix subscript, such as LET A (-1) = 0*
- b. an unreasonably large matrix subscript: X=A(65536)*
- c. LOG-negative or zero argument, as LOG(-X)**
- d. SQR-negative argument, as SQR(-4)
- e. At B if A is a negative value and B is not an integer.

(It works if a constant is used instead of a variable, i.e., -4t B, because exponentiation is performed before unary minus.)

- f. A call to USR before the address of the machine language subroutine has been patched in.

Error Message	What caused the error and how to fix it
---------------	---

**Subscripts must be equal to or greater than 0, and less than or equal to 65535. ** Be sure the argument is within the range of the function being used.*

NEXT WITHOUT FOR	The variable in a NEXT statement corresponds to no previously executed FOR statement.
-------------------------	---

The FOR part of a FOR ... NEXT loop must be inserted or the offending NEXT part of the loop must be deleted. Be sure the index variables are the same at both ends of the loop.

Example: FOR I = 1 TO 10
NEXT I

OUT OF DATA A READ	A statement was executed but all of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program.
---------------------------	--

Use the RESTORE: statement to restore the data so PET can read it again, or restrict the number of READs to the correct number of DATA elements, or add more DATA elements, or use a flag at end of data list check for it before reading.

OVERFLOW	The result of a calculation was too large to be represented in BASIC's number format. (If an underflow occurs, zero is given as the result and execution continues without any error message being printed.)
-----------------	--

You requested a number greater than even PET can remember. Try asking for a smaller number. The largest possible number is 1.70141 183E+38. Change the order of your calculations.

REDIMENSIONED ARRAY	After a matrix was dimensioned, another dimension statement for the same matrix was encountered. This error often occurs if a matrix has been given the default dimension 10 because a statement like A(1)=3 is encountered and then later in the program, a DIM A(100)
----------------------------	---

is found.

 www.commodore.ca

Free for personal use but you must have written permission to reproduce

Check to see if you have used a GOTO to branch back to a statement preceding the DIM statement, or see if the DIM statement is inside a FOR . . . NEXT loop or a subroutine that will be executed more than once, or if you have used an array element before using the DIM statement. Make DIM one of the first lines in your program.

RETURN WITHOUT GOSUB A RETURN statement was encountered without a previous GOSUB statement being executed.

Either insert a GOSUB or delete the RETURN. Maybe you fell through the program and should enter an END statement before the first subroutine to prevent falling through.

STRING FORMULA TOO COMPLEX A string expression was too complex.

Break up the string into two or more shorter expressions.

STRING TOO LONG Attempt was made by use of the concatenation operator to create a string more than 255 characters long. Note that a number is printed as SPACE-NUMBER-CURSOR RIGHT.

SUBSCRIPT OUT OF RANGE An attempt was made to reference a matrix element which is outside the dimensions of the matrix.

This error can occur if the wrong number of dimensions is used in a matrix reference; for instance, LET A(1,1,1)=Z when A has been dimensioned **DIM A(2,2)**.

You must either increase the space you requested for the array (change a DIM A(10) to a DIM A(20), for example) or alter the number of dimensions you asked for (change from DIM A(10,10) to DIM A(10,10,10) or from DIM B(10,10,10) to DIM B(10,10) for example).

SYNTAX ERROR Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.

This one is hard to find, but easy to fix. Examine the offending statement carefully and insert or delete whatever is necessary.

TYPE MISMATCH The left-hand side of an assignment statement was a numeric variable and the right-hand side was a string, or vice versa; or a function which expected a string argument was given a numeric one or vice versa.

Can't mix statement types, so change one side of the assignment statement so it agrees with the other side (sides meet at the = sign). Check the function argument types and use the correct type (numeric or string).

UNDEFINED STATEMENT An attempt was made to GOTO, GOSUB or THEN to a statement which does not exist.

Insert the necessary statement number or branch to another statement number.

UNDEFINED USER FUNCTION Reference was made to a user-defined function which had never been defined.

Define the function.

FILE OPEN You have attempted to open a previously opened file. Check logical file numbers (1st parameter in the OPEN statement) and be sure you use unique numbers for each file or close the file. File is now closed.

FILE NOT OPEN You have attempted to read from, write to, or close a file not previously opened.

Open the file.

NOT INPUT FILE You tried to INPUT# from a file opened for writing. Reading requires a Q as the 3rd parameter of the OPEN statement. Read (0) is the default option.

NOT OUTPUT FILE You tried to PRINT# to a file opened for reading.

Writing to a file requires a I (or a 2 if you want an EOT at the end of the file) as the 3rd parameter in the OPEN statement.

DEVICE NOT PRESENT You have attempted to open a file on a device which is 'invisible' to PET.

Check device numbers (2nd parameter in the OPEN statement) and be sure the device is assigned and connected properly and turned on.

2. Basic commands

Basic Commands and Statements

COMMAND/ STATEMENT	EXAMPLE	PURPOSE
CLR	CLR	Sets variables to zero or null.
CMD	CMD D	Keep IEEE device D open to monitor bus.
CONT	CONT	Continue program execution after a STOP command. No program changes permitted.
GOTO	GOTO L	Continue program execution at line L after a STOP command. Program changes are permitted.
FRE	PRINT FIRE (0)	Returns number of bytes of available memory.
END	END	Ends program.
LET	LET A=2	Assign a value to a variable.
LIST	LIST LIST -L LIST L-M LIST L	Lists current program. Lists current program through line L. Lists lines L through M of current program. Lists current program from line L to end.
LOAD	LOAD LOAD "NAME" LOAD "NAME," D	Loads next encountered program from built-in tape unit. Loads program NAME from built-in tape unit. Loads program NAME from device D.
NEW	NEW	Deletes current program from memory, sets variables to zero.
PEEK	PEEK(A)	Returns byte value from address A.
POKE	POKE A,B	Loads byte B into address A.
PRINT	PRINT A PRINT A\$ PRINT #L,A PRINT #L,A\$	Prints value of A on display screen. Prints specified string on screen. Prints value of A on logical file L. Prints specified string on logical file L.
RUN	RUN RUN L	Begins execution of program at lowest line number. Begins execution of program at line L.
SAVE	SAVE SAVE "NAME" SAVE "NAME,"D SAVE "NAME,"D,C	Saves current program on built-in tape unit. Saves current program NAME on built-in tape unit. Saves current program NAME on device D. Saves program NAME on device D. C specifies EOF or EOT.
STOP	STOP	Stops program execution.
SYS	SYS(X)	Complete control of PET is transferred to a subsystem at decimal address contained in the argument.

Basic Commands and Statements (Continued)

COMMAND/ STATEMENT	EXAMPLE	PURPOSE
TI\$ TI	TI\$="HHMMSS" PRINT TI	Sets PET's internal clock to real time. Displays number of 'jiffies' since PET was powered up or clock was zeroed. (A jiffy - 1 r'60 of a second.)
USR	USR(X)	Transfers program control to a program whose address is at locations 1 and 2. X is a parameter passed to and from the machine language program.
WAIT	WAIT A,B,C <i>(Note: the WAIT command may not operate on your particular PET)</i>	Stops execution of BASIC until contents of location A, ANDed with B and exclusive ORed with C, is not equal to zero. C is optional and defaults to zero.
CLOSE	10 CLOSE L	Closes logical file L.
DATA	10 DATA 1,2,3,4 20 DATA TOM,SUE 30 DATA "TOM DOE"	Specifies data to be read from left to right. Alphabets do not need to be enclosed in quotes. If strings contain spaces, commas, colons, or graphic characters, the string must be enclosed in quotes.
DIM	10 DIM A(n) 20 DIM A(n,m,o,p) 30 DIM A(o),B(m) 40 DIM Aft 50 DIM A\$(n)	Specifies maximum number of elements in an array or matrix. Specifies maximum number of dimensions in an array. Number of arrays limited by memory. May be dimensioned dynamically. Strings may be dimensioned.
END	999 END	Terminates program execution.
GET	10 GET C 20 GET C\$ 30 GET #L,C 40 GET #L,C\$	Accepts single numeric character from keyboard. Accepts single string character from keyboard. Accepts single character from specified logical file. Accepts specified single string character from logical file.
INPUT	10 INPUT A 20 INPUT A\$ 30 INPUT A,A\$,B,B\$ 40 INPUT #L,A 50 INPUT #L,A\$ 60 INPUT #L,A,A\$,B,B\$	Accepts value of A from keyboard. Accepts value of string variable A from keyboard. The string does not have to be enclosed in quotes. Accepts specified values from keyboard. Accepts value of A from logical file L. Accepts specified string from logical file L. Accepts specified values and strings from logical file L. Strings do not have to be enclosed in quotes.

Basic Commands and Statements (Continued)

COMMAND; STATEMENT	EXAMPLE	PURPOSE
LOAD	10 LOAD 20 LOAD "NAME" 30 LOAD "NAME",D	Loads next encountered program or file, on built-in tape unit, into PET's memory. Loads program or file NAME into memory from built-in tape unit. Loads specified file NAME from device D.
OPEN	10 OPEN L 20 OPEN L,D 30 OPEN L,D,C 40 OPEN L,D,C,"NAME"	Opens logical file L for read only from built-in tape unit. Opens logical file L for read only from device D. Opens logical file L for command C from device D. Opens logical file L on device D. If device D accepts formatted files, file NAME is positioned for command.
POS	10 PRINT POS(0)	Prints next available print position (position of cursor on screen I.
PRINT	10 PRINT A 20 PRINT AS 30 PRINT A,A\$ 40 PRINT A;A\$ 50 PRINT #L,A 60 PRINT #L,A\$	Prints value of A on display screen, Prints specified string on screen. Prints specified values or strings on screen, beginning in next available print position (pre-TABbed positions are in columns 10,20,30,40, etc.). Prints on specified values and strings on screen separated by 3 spaces if numeric, concatenated if string. Prints specified value on logical file L. Prints specified string on logical file L.
READ	10 READ A 20 READ AS 30 READ A,A\$,B,B\$	Obtains value of A from a DATA statement. Obtains string value of A from a DATA statement. Obtains specified values for strings and numeric variables from DATA statements.
REM	10 REM "COMMENT"	Inserts non-executable comments in a program for documentation purposes.
RESTORE	10 RESTORE	Permits re-reading of DATA statements without re-running program.
TAB	10 PRINT TAB(N);A 20 PRINT TAB(N);A\$	Prints value of A in character position N+1 on screen. Prints string beginning in character position N+1 on screen.
VERIFY	1 2 ERIFY 20 VERIFY "NAME"	Verifies most recent program saved on built-in cassette by reading it and comparing it with program still in PET's memory. Verifies specified program NAME saved on built-in cassette by reading it and comparing it with program still in PET's memory.

Basic Commands and Statements (Continued)

COMMAND; STATEMENT	EXAMPLE	PURPOSE
	30 VERIFY "NAME",D	Verifies specified program NAME saved on device D by reading it and comparing it with program still in PET's memory.
SPC	10 SPC(N)	Prints N spaces or blanks.
FOR . . . NEXT	10 FOR A = 1 TO 20 90 NEXT A	Loop control. Performs all instructions between FOR and NEXT as many times as specified by index. In this example, the index variable is A.
STEP	10 FOR A = 1 TO 20 STEP 2 90 NEXT A	Step specifies size of increment to be added to index to increase or decrease its value towards the desired number of iterations.
IF ... THEN	10 IF A = 10 THEN PRINT A	If condition is 'TRUE,' instruction following 'THEN' (in this example, 'PRINT A') would be executed. Otherwise, the next statement in sequence is executed.
IF ... GOTO	10 IF A=1 GOTO L	If condition is true, control is transferred to specified line. Otherwise, the next statement, following the) F . . GOTO, is executed
GOTO	10 GOTO L	Transfers control (jumps) to specified line, skipping over intervening lines.
GOSUB	10 GOSUB L	Begins execution of a subroutine which begins on a specified line.
ON ... GOTO	10 ON A GOTO L,M,N	Transfers control to specified line (in this example, L,M, or N, depending on value of index A.
ON... GOSUB	10 ON A GOSUB L,M,N	Begins execution of subroutine which begins on line L,M, or N, depending on the value of index A.
RETURN	9990 RETURN	Subroutine exit; transfers control to the statement following most recent GOSUB directing transfer to the subroutine.

String Functions

FUNCTION	EXAMPLE	PURPOSE
ASC	10 A=ASC("XYZ")	Returns integer value corresponding to ASCII code of first character in string.
CHRS	10 A\$=CHRS(N)	Returns character corresponding to ASCII code number.
LEFT\$	10 ?LEFT\$(X\$,A)	Returns leftmost A characters from string.
LEN	10 ?LEN(X\$)	Returns length of string.

String Functions (Continued)		
FUNCTION	EXAMPLE	PURPOSE
MID\$	10 ?MID\$(X\$,A,B)	Returns B characters from string, starting with the Ath character.
RIGHTS	10 ?RIGHT\$(X\$,A)	Returns rightmost A characters from string.
S 1 R\$	10 A\$=STR\$(A)	Returns string representation of number.
VAL	10 A=VAL(A\$) 20 A=VAL("A")	Returns numeric representation of string. If string not numeric, returns "0".
ASC, LEN and VAL functions return numerical results. They may be used as part of an expression. Assignment statements are used here for examples only; other statement types may be used.		

Arithmetic Functions

FUNCTION	EXAMPLE	PURPOSE
ABS	10 C=ABS(A)	Returns magnitude of argument without regard to sign.
ATN	10 C=ATN(A)	Returns arctangent of argument. C will be expressed in radians.
COS	10 C=COS(A)	Returns cosine of argument. A must be expressed in radians.
DEF FN	10 DEF FNA(B)-C*D	Allows user to define a function. Function label A must be a single letter; argument B a dummy.
EXP	10 C=EXP(A)	Returns constant 'e' raised to power of the argument. In this example, eA.
INT	10 C=INT(A)	Returns largest integer less than or equal to argument.
LOG	10 C=LOG(A)	Returns natural logarithm of argument. Argument must be greater than or equal to zero.
RND	10 C=RND(A)	Generates a random number between zero and one. If A is less than 0, the same negative random number is produced in each call to RND. If A = 0, the same positive sequence of random numbers is generated each time RND is called. If A is greater than 0, a new sequence is produced for each call to RND.
SGN	10 C=SGN(A)	Returns -1 if argument is negative, returns 0 if argument is zero, and returns +1 if argument is positive.
SIN	10 C=SIN(A)	Returns sine of argument. A must be expressed in radians.
SQR	10 C=SQR(A)	Returns square root of argument.
TAN	10 C=TAN(A)	Returns tangent of argument. A must be expressed in radians.

Arithmetic Operators		
SYMBOL	EXAMPLE	PURPOSE
	10 A=B 20 LET A=B	Assigns a value to a variable. Let is optional.
	30 PRINT A 2	Exponentiation; in example, A ² .
/	35 C=A/8	Division
	40 C=A*8	Multiplication
	50 C=A+B	Addition
	60 C=A-8	Subtraction
	10 IF A=B THEN PRINT C	A 'equals' B.
<>	10 IF A<-B THEN C=4	A 'does not equal' B.
	10 IF A<B THEN C\$="X"	A 'is less than' B.
	10 IF A>B THEN C\$=D\$+E\$	A 'is greater than' B.
	10 IF A<=B THEN C=20	A 'is less than or equal to' B.
	10 IF A<=B THEN C=D-1	A 'is greater than or equal to' B.
AND	10 IF A AND B THEN C=0	A and B must BOTH be true for statement 10 to be true.
OR	20 IF A OR B THEN C=90	A must be true or B must be true for statement 20 to be true.
NOT	30 IF NOT A THEN PRINT C	Expression is true if A is false.

"NOTE The numerical values used in the evaluation of logical comparisons are: 'TRUE' is any non-zero number and 'FALSE' is zero.

Special Symbols, Commands and Statements

SYMBOLS, COMMANDS, STATEMENTS	EXAMPLE	PURPOSE
	10 A=1:8=2:C=3	Allows multiple statements on a line.
	10 PRINT A;B	Allows same line printing. Elements are separated by a space.
	20 PRINT A\$;B\$	Allows same line printing. String elements are concatenated.
	10 PRINT A,B	Allows same line printing. Elements are separated and printed in pre-TABbed print positions (columns 11,21,31, etc.)
	LOAD "NAME",D	Separates elements in LOAD, SAVE, OPEN, and VERIFY.
?	10 ?A	Abbreviation for PRINT. Stores as one character; lists as word PRINT. Do not use for PRINT #.

Special Symbols, Commands and Statements (Continued)

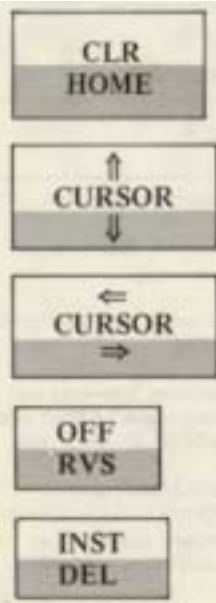
SYMBOLS, COMMANDS, STATEMENTS	EXAMPLE	PURPOSE
\$	10 A\$="ABCDEFGH"	String identifier.
%	10 A%=INT(X)	Integer
'	10 A\$="ABCDEF"	String enclosures.
␣		carriage return Must follow every command, statement, or data entry; causes cursor to return to leftmost position on next lowest line. Signals "END OF INPUT LINE."
TT(p)		Value of Pi: 3.1415927.

I/O Commands

SYMBOL	COMMAND	PURPOSE
L=	1-255	
C=	0: READ OPEN L,D,C	Note: PET will not read past C=1: WRITE an EOT (end of tape) marker.
C=	2: WRITE AND PUT	EOT at end of file.
D=	1 CASSETTE	
D=	2 2ND CASSETTE	
D=	4-15 IEEE BUSS	

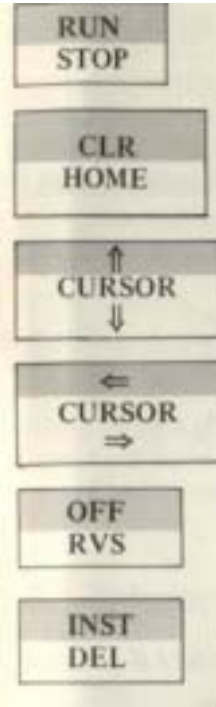
3. Special keys

The following keys, when pressed while the **SHIFT** key is being held down, will perform the following functions:



- LOADS** and **RUNS** the next encountered program from the built-in tape unit.
- CLR HOME**: Clears print from screen and moves cursor to upper left corner of screen. Program statements and all variables are retained.
- CURSOR** (up/down arrows): Moves cursor one space up. Will not scroll off top of screen. Does not delete characters as it passes over them.
- CURSOR** (left/right arrows): Moves cursor one space left (backspace). Wraps around to rightmost position on next highest line. Does not delete characters as it passes over them.
- OFF RVS**: Resets reverse field printing to normal printing.
- INST DEL**: Inserts a space immediately in cursor position. All characters to right of inserted space are moved one space to right. Stops when 80th character is filled.

When the **SHIFT** key is not pressed, the keys will perform different functions, as indicated:



- RUN STOP**: Stops execution of command in progress (LIST, LOAD, RUN, etc.).
- CLR HOME**: Returns cursor to upper left corner of screen.
- CURSOR** (up/down arrows): Moves cursor one space down. When cursor is at bottom of screen, print will scroll off top of screen. Does not delete characters as it passes over them.
- CURSOR** (left/right arrows): Moves cursor one space right. Will wrap around to left most position of next lowest line. Does not delete characters as it passes over them.
- OFF RVS**: Enables reverse field print (black characters on a white background).
- INST DEL**: Deletes character immediately to left of cursor. All characters to right of deletion are moved one space left. Line is filled with trailing blanks if needed.

4. Cleaning your PET

- With power switch in "OFF" position, gently wipe keytops with a slightly damp cloth. Do not flood with water.
- Use any of the available lens-cleaner sprays to clean the video screen. Spray screen lightly, and dry with a soft, non-linting cloth or tissue.
- Wipe the cabinet with a soaped, well-wrung sponge. Do not use any commercial abrasive cleaners. Rinse by wiping with clean, slightly damp sponge or soft cloth. Do not immerse in water.
- Clean the recorder unit inside by touching dust particles with slightly damp cloth or sponge. Do not wipe surface ... cloth will snag on metal or plastic parts and may cause breakage. (See next section for more detailed instructions.)
- The outside of the recorder unit may be cleaned in the same manner as the keytops.

5. Cleaning and demagnetizing your tape deck head

To be performed every 50-100 hours of tape running time or when cassette unit fails to read tapes reliably.

You'll need the following tools and materials:

- 1) Tape head cleaner. ("NORTRONICS" Brand is recommended.) Do not use Trichloroethane or any other plastic or rubber solvent. Alcohol may be used in an emergency, but is not recommended for long term use.

Cotton Swabs. "Johnson & Johnson" Brand is recommended: the cotton seems to stick to the end of the swab better.
- 3) Tape Head Demagnetizer: "NORTRONICS," "HAND-DE-MAG" and "ROBINS" brands are recommended. Unit must have protective plastic or rubber covering on pole piece so as not to scratch delicate head gap.

HOW TO PROCEED:

- 1) Turn Off PL T.
- 2) Press PLAY on tape deck to make heads available.
- 3) Use tape head cleaner and one side of a cotton swab to clean surfaces of RECORD/PLAY (RIP) and erase head. (See Figure 7.)

Scrub gently, noting if there is any build-up of tape oxide particles on or around head gap of the R/P head. If so, this is sufficient reason for unreliable performance.

Also clean pinch roller and other tape bearing surfaces if tape head cleaner is suitable for this purpose. (Check label.)
- 4) Plug in demagnetizer and activate it while it is at least one foot away from cassette heads.
- 5) Slowly move demagnetizer up to RIP head and around on head surface. Rate of motion should be approximately one inch per second during this time.
- 6) Slowly move demagnetizer to erase head and then to all other ferrous metal surfaces which come into proximity with the tape.
- 7) Now slowly move demagnetizer away from heads and do not

de-activate field until demagnetizer is at least two feet away from heads.

Tape head cleaning and demagnetizer procedure is now complete. Inspect R/P head surface for wear. If tape has worn a groove on head surface more than a Couple of tape thicknesses deep and program reading performance is still poor, then replacement of tape head is indicated. (This usually occurs after three thousand or more hours of tape running time.)

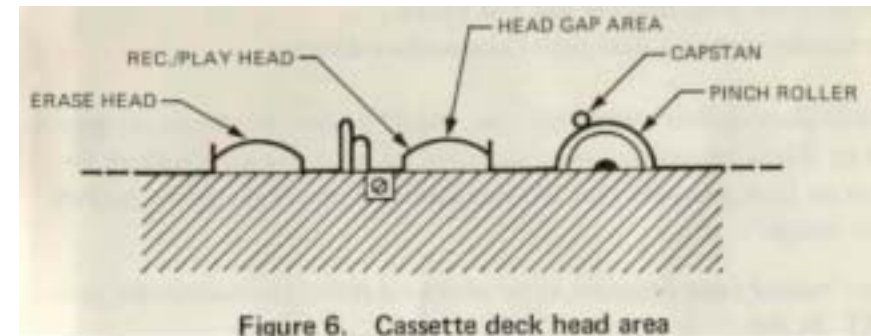


Figure 6. Cassette deck head area

6. References

- Entering BASIC. J. Sack and J. Meadows, Science Research Associates, 1973.
- BASIC: A Computer Programming Language, C. Pegels, Holden-Day, Inc., 1973.
- BASIC Programming. J. Kemeny and T. Kurtz, Peoples Computer Co., 1010 Doyle (P.O. Box 310), Menlo Park, CA 94025, 1967.
- BASIC, Albrecht, Finkle and Brown, Peoples Computer Co., 1010 Doyle (P.O. Box 310), Menlo Park, CA 94025, 1973.
- A Guided Tour of Computer Programming in BASIC, T. Dwyer, Houghton Mifflin Co., 1973.
- Programming Time Shared Computer in BASIC, Eugene H. Barnett, Wiley-Interscience. L/C 72-175789 (\$12.00).
- Programming Language #2, Digital Equipment Corp., Maynard, MA 0101 BASIC Computer Games, Software Distribution Center, Digital Equipment Corp., Maynard. MA 01754 (\$7.50).
- What to Do After You Hit Return, Peoples Computer Co., 1010 Doyle (P.O. Box 310), Menlo Park, CA 94025 (\$6.95).
- Basic BASIC, James S. Coan, Hayden Book Co., Rochelle Park, N.J.
- Advanced BASIC, James S. Coan, Hayden Book Co., Rochelle Park, N.J.

AN IMPORTANT NOTE TO ALL PET OWNERS

Create PET programs for fun and profit ...

But make sure you deal with Commodore directly.

We at Commodore encourage you, the PET user, to submit programs to us. These programs, when accepted by us, will earn royalties for you on each program sold. So you can now create programs for fun and profit,

Just submit your program, after checking it out thoroughly on your PET, to the

Software Program Manager

901 California Avenue

Palo Alto. CA 94304

This is the only address if you want Commodore to evaluate and distribute your program. It's been brought to our attention that some free-lance operators are soliciting PET users for programs and implying that they represent Commodore,

Absolutely not so! Commodore is not represented by any user club or newspaper. The only address for Commodore PET program information or sales in the United States is Commodore at 901 California Avenue in Palo Alto.

So, protect yourself. We want your programs. And we know you want to make sure you're dealing with Commodore directly.



Free for personal use but you must have written permission to reproduce

Commodore **Business Machines, Inc..**
901 California Avenue Palo Alto,
California 94304, USA

Commodore/MOS

Valley Forge Corporate Center
950 Rittenhouse Road
Norristown, Pennsylvania 19401, USA

Commodore Business Machines Limited
3370 Pharmacy Avenue Agincourt,
Ontario, Canada M1W2K4

Commodore Business Machines Limited
Eaglescliffe Industrial Estate Eaglescliffe,
Stockton on Tees Teeside TS 160 PN,
England

Commodore Business Machines Limited
360 Euston Road

London NW1 3BL, England

Commodore Buromaschinen GmbH

Frankfurter Strasse 171-175 6078
Neu Isenburg West Germany

Commodore Japan Limited
Taisei-Denshi Building 8-14
Ikue 1-Chome Asahi-Ku,
Osaka 535, Japan

Commodore Electronics (Hong Kong) Ltd

Watsons Estates
Block C, 11th floor
Hong Kong, Hong Kong

